BAB II
LANDASAN TEORI

2.1. Pengertian Penjadwalan

Penjadwalan merupakan pengalokasian sumber daya dengan mengurutkan pembuatan produk selama periode waktu tertentu. Optimalitas dalam sebuah pekerjaan adalah tujuan dari adanya penjadwalan. Umumnya, penjadwalan dilakukan dengan rencana pengaturan urutan kerja pada setiap pusat pemrosesan. Menurut Ginting (2009), penjadwalan adalah pengurutan pembuatan/pengerjaan produk secara menyeluruh yang dikerjakan pada beberapa buah mesin.

Menurut Masudin, Utama, and Susastro (2014), adanya penjadwalan produksi yang sistematis dapat memberikan kepuasan terhadap pelanggan, salah satunya dengan penyelesaian pesanan yang tepat waktu. Waktu proses dan *job* merupakan elemen dasar dari penjadwalan. Istilah *job* dalam penjadwalan merupakan aktivitas/ operasi. Selain itu, pada setiap operasi membutuhkan alokasi tenaga kerja dan mesin peralatan produksi. Fungsi dilakukannya penjadwalan yaitu membantu dalam pengambilan keputusan, serta agar kelangsungan produksi lebih efisien.

2.2. Tujuan Penjadwalan

Proses penjadwalan memiliki beberapa tujuan. Tujuan hasil dari penjadwalan yaitu mendapatkan hasil yang lebih baik sesuai dengan harapan. Bedworth and Bailey (1999), mendeskripsikan beberapa tujuan dari adanya aktivitas penjadwalan sebagai berikut:

a) Meningkatkan penggunaan sumberdaya atau mengurangi waktu tunggu. Sehingga total waktu proses saat produksi dapat berkurang, serta produktivitas dapat meningkat sesuai dengan yang diharapkan.
b) Mengurangi *job* yang menunggu dalam antrian ketika sumberdaya yang ada masih mengerjakan *job* yang lain.
c) Mengurangi keterlambatan pada *job* yang memiliki batas waktu penyelesaian, sehingga dapat meminimasi biaya denda (*penalty cost*)

d) Membantu dalam pengambilan keputusan tentang perencanaan kapasitas pabrik dan jenis kapasitas yang dibutuhkan, sehingga penambahan biaya yang mahal dapat dihindarkan.

2.3. Penjadwalan *Flowshop*

Penjadwalan *flowshop* adalah sistem pemrosesan dimana urutan setiap pekerjaan sepenuhnya ditentukan, serta setiap pekerjaan mengunjungi stasiun kerja dalam urutan yang sama. Penjadwalan flowshop memiliki aturan yaitu suatu pekerjaan akan mengalir berdasarkan produk dan tidak akan kembali pada tahap sebelumnya. Dengan demikian, menurut M. Firdaus, I. Masudin, and D. M. Utama (2015) setiap stasiun kerja diberi nomor 1, 2, ..., m, dan setiap pekerjaan yang mengunjungi stasiun kerja tersebut dalam urutan angka (J_1, J_2, ..., n). Produk dengan desain yang stabil dan diproduksi dalam jumlah banyak dapat menerapkan penjadwalan *flowshop*.

Karakteristik dasar pada penjadwalan *flowshop* sebagai berikut:

a) Terdapat n *job* yang tersedia dan siap untuk diproses pada waktu t = 0.
b) Terdapat m mesin berbeda yang tersedia secara kontinu.
c) Waktu *set up independent* terhadap urutan pengerjaan.
d) Operasi-operasi individual tidak dapat dipecah-pecah.

Menurut Baker and Trietsch (2009), *flowshop* terbagi menjadi beberapa bagian berdasarkan aliran produksinya:

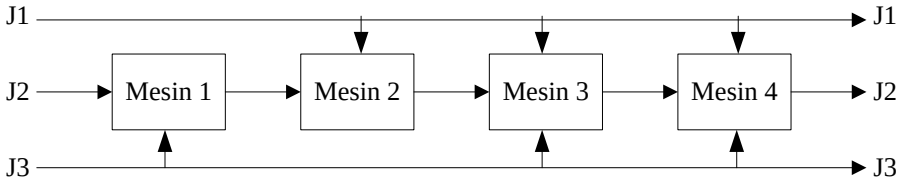
a) *Pure Flowshop*

Yakni semua jenis pekerjaan memiliki pola aliran proses yang identik. Setiap pekerjaan akan melewati seluruh mesin yang bekerja dari proses awal hingga akhir sesuai dengan urutan prosesnya.

Gambar 2. 1 Aliran *Pure Flowshop*

b) *Skip Flowshop*

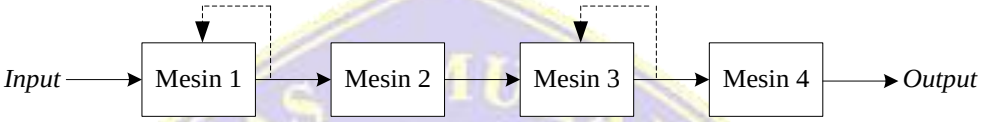
Pola aliran untuk setiap pekerjaan melalui urutan proses yang sama, tetapi ada beberapa pekerjaan yang tidak diproses pada mesin-mesin tertentu.



Gambar 2. 2 Aliran *Skip Flowshop*

c) *Re-entrant Flowshop*

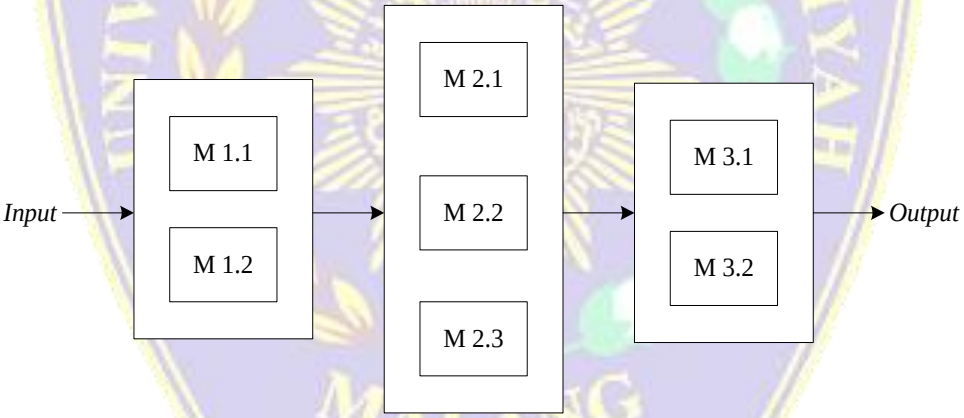
Yakni pola aliran proses yang menggunakan satu atau beberapa mesin lebih dari sekali dalam membuat produk.



Gambar 2. 3 Aliran *Re-entrant Flowshop*

d) *Compound Flowshop*

Yakni pola aliran proses yang memuat kelompok jenis mesin pada setiap tahap prosesnya. Kelompok mesin ini biasanya berupa mesin paralel.



Gambar 2. 4 Aliran Compound Flowshop

1.4. Klasifikasi Penjadwalan

Dana Marsetiya Utama (2019) mengklasifikasikan penjadwalan produksi menjadi:

a) Penjadwalan Mesin Tunggal (*single machine*)

Penjadwalan mesin tunggal merupakan masalah penjadwalan yang mempunyai sumber tunggal. Semua waktu proses pada penjadwalan mesin tunggal bersifat deterministik. Hasil yang diperoleh dari penelitian model

mesin tunggal tidak hanya menyampaikan wawasan pada lingkungan mesin tunggal saja, namun juga dapat menjadikan dasar pemikiran untuk heuristik pada lingkungan mesin yang lebih rumit. Pada kenyataannya, masalah penjadwalan yang rumit sering diuraikan menjadi bagian dari masalah yang berhubungan dengan mesin tunggal.

b) Penjadwalan Mesin Paralel

Penjadwalan paralel terbagi menjadi:

1. Prinsip penjadwalan n *job* terhadap mesin parallel identik (*identical machine parallel*) adalah pengalokasian beban ke mesin yang lebih dahulu kosong/idle.
2. Penjadwalan n job terhadap mesin paralel *non identic* yaitu setiap mesin mempunyai fungsi yang sama, namun waktu prosesnya berbeda. *Flow time* tidak dapat dievaluasi langsung dari waktu proses. Tidak selalu alternatif waktu terpendek dari setiap *job* akan menjadi keputusan alokasi pada mesin.
3. Penjadwalan n job pada mesin paralel *unrelated* perluasan dari paralel *nonidentic*. Terdapat m mesin paralel, dimana mesin i untuk memproses *job* maka kecepatan mesin adalah vij.

1.5. Istilah Dalam Penjadwalan

Menurut Ginting (2009) istilah umum yang sering dijumpai dalam penjadwalan sebagai berikut:

a) *Processing Time* / Waktu Proses (t_i), yaitu waktu yang diperlukan untuk memberikan nilai tambah pada *job* ke-i. Waktu untuk persiapan dan waktu untuk pengaturan proses sudah termasuk dalam waktu proses ini.

b) *Completion Time* / Waktu Penyelesaian (C_i), yaitu waktu yang diperlukan untuk bisa menyelesaikan *job* pertama dimulai (t=0) sampai pada *job* ke-i selesai dikerjakan.

c) *Makespan* (M), yaitu total waktu penyelesaian *job*, mulai dari urutan *job* pertama yang dikerjakan pada mesin pertama sampai kepada *job* terakhir pada mesin terakhir juga.

d) *Flow Time* (F_i), yaitu rentang waktu yang dibutuhkan mulai dari job masuk ke dalam suatu tahap proses sampai job yang bersangkutanselesai dikerjakan. Dengan kata lain, *flow time* sama dengan *processing time* ditambah dengan waktu ketika *job* menunggu sebelum diproses.

e) *Ready Time* (R_i)

Menunjukkan saat *job* ke-i siap dikerjakan (dijadwalkan)

1.6. Klasifikasi Kondisi Penjadwalan

Klasifikasi penjadwalan produksi dapat diketahui dari perbedaan kondisi yang mendasarinya. Baker and Trietsch (2009) mengklasifikasikan perbedaan kondisi yang sering terjadi dalam proses produksi :

1. Berdasarkan pola kedatangan *job* terbagi menjadi dua bagian yaitu:
   a) Statik, semua *job* datang secara bersamaan dan siap dikerjakan pada mesin yang telah tersedia atau menganggur.
   b) Dinamik, kedatangan *job* tidak menentu dan *job* selalu di*update* jika ada *job* baru yang masuk.
2. Berdasarkan waktu proses terbagi menjadi dua bagian yaitu:
   a) Deterministik, waktu proses dari *job* yang diterima sudah diketahui dengan pasti.
   b) Stokastik, waktu proses dari *job* yang diterima belum pasti, sehingga perlu diperkirakan dengan distribusi probabilitas.

2.7. Prioritas Dispatching Rules

Aturan prioritas digunakan untuk mengetahui pekerjaan mana yang akan dikerjakan terlebih dahulu. Istilah lain dari dispatching rule yaitu *scheduling in advance*. Menurut Baker and Trietsch, (2009) *scheduling in advance* merupakan teknik *heuristic* yang sering digunakan dalam penjadwalan. Metode yang sering digunakan perusahaan adalah aturan *First Come First Serve* (FCFS). Pengurutan penjadwalan aturan FCFS dilakukan berdasarkan waktu kedatangan atau pesanan pelanggan. Setiap pekerjaan yang datang pertama pada suatu pusat kerja yang diproses terlebih dahulu. Aturan FCFS sering digunakan pada bidang jasa seperti bank, kantor pos, bengkel, dan sebagainya.

2.8. Ukuran Performansi Penjadwalan

Menurut Baker and Trietsch (2009), ukuran performansi digunakan untuk mengevaluasi hasil penjadwalan dalam pengambilan keputusan. Berikut ini adalah ukuran performansi yang terbagi menjadi dua bagian yaitu:

1. Ukuran performansi penjadwalan berdasarkan atribut *job*.
   a) *Flow time* (F_i)

      Flow time adalah rentang waktu yang dibutuhkan oleh suatu *job* dari saat job tersebut masuk kedalam tahap proses sampai job tersebut selesai.

      $F_i = t_i + w_i$ ……(1)

      Dimana F_i : *Flowtime* untuk *job* ke-i

      t_i : waktu proses untuk *job* ke-i

      w_i : waktu tunggu sebelum diproses untuk *job* ke-i

   b) *Completion time* (C_i)

      Completion time adalah waktu yang diperlukan untuk bisa menyelesaikan *job* pertama dimulai (t=0) sampai pada *job* ke-i selesai dikerjakan.

      $C_i = F_i + r_i$ ……(2)

      Dimana C_i : *Completion time* untuk *job* ke-i

      F_i : *Flowtime* untuk *job* ke-i

      r_i : waktu ketika *job* ke-i selesai

   c) *Mean flowtime*

      Mean flow time adalah rata-rata waktu yang dihabiskan untuk semua *job* melewati seluruh urutan proses produksi.

      $F = \frac{1}{n}\sum_{i=1}^{n} F_i$ ……(3)

      Dimana F : *Mean flowtime*

      F_i : *Flowtime* untuk *job* ke-i

2. Ukuran performansi penjadwalan berdasarkan atribut pabrik.
   a) Minimasi *makespan*

      Minimasi *makespan* adalah jangka waktu penyelesaian semua *job* yang akan dijadwalkan yang merupakan jumlah dari seluruh proses.

      $M_s = \sum_{t=1}^{n} t_i$ ……(4)

      Dimana M_s : *Makespan*

t_i : Waktu proses untuk *job* ke-i

2.9. *Gantt Chart* Minimasi Makespan

Menurut Ginting (2009), *Gantt chart* merupakan representasi grafis hubungan antara alokasi sumber daya dengan waktu. Peta *gantt chart* digambarkan dalam bentuk batang dan analog. Pada *gantt chart* sumbu vertikal menggambarkan sumber daya yang digunakan, sedangkan sumbu horizontal menggambarkan satuan waktu. G*antt chart* dikenal sebagai alat bantu yang sederhana dan mudah untuk dibaca dalam menyelesaikan masalah penjadwalan. *Gantt chart* dapat membantu penggunanya untuk memastikan bahwa,

a) Urutan kinerja telah diperhitungkan
b) Perkiraan waktu kegiatan telah tercatat
c) Semua kegiatan telah direncanakan
d) Keseluruhan waktu proyek telah dibuat

Gantt chart tidak bisa menunjukkan keterkaitan antara aktivitas dan bagaimana akibat dari aktivitas satu dengan aktivitas lain bila waktunya terlambat atau dipercepat, sehingga modifikasi terhadap *gantt chart* perlu dilakukan. Untuk bisa mengatasi kekurangan dari *gantt chart*, maka dikembangkan sebuah teknik baru yaitu jaringan (*network*).

Menurut Abdel-Basset et al. (2018), *Gantt chart* dengan tujuan meminimasi makespan ditunjukkan oleh persamaan berikut, serta gambar 2.5 menunjukkan contoh *gantt chart*:

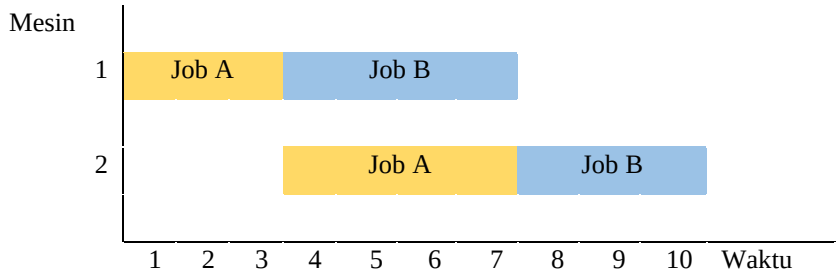
$c(j_1, i_1) = t_{j_1,i_1}$, k = 1, h = 1……(5)

$c(j_1, i_h) = c(j_1, i_{h-1}) + t_{j_1,i_h}$, h = 2, … , m……(6)

$c(j_k, i_1) = c(j_{k-1}, i_1) + t_{j_k,i_1 i_1}$, k = 2, … , n……..(7)

$c(j_k, i_h) = \max(c(j_{k-1}, i_h), c(j_k, i_{h-1}))$ + t_{j_k,i_h}, k = 2, … , n, h = 2, … , m..(8)

c* = $c(j_n, i_m)$……..……..…(9)



Gambar 2. 5 Contoh *Gantt Chart*

Kelebihan dari *gantt chart* adalah:

1. Dapat menunjukkan urusan kegiatan, kegiatan dan waktu.
2. Jika jumlah kegiatan tidak banyak atau hanya sekedar jadwal induk, maka alat bantu *gantt chart* menjadi pilihan pertama dalam proses perencanaan dan pengendalian kegiatan.

Selain kelebihan, berikut adalah kekurangan dari *gantt chart*:

1. Tidak memperhatikan saling ketergantungan dan hubungan antar kegiatan, sehingga sulit untuk diantisipasi jika terjadi keterlambatan suatu kegiatan terhadap keseluruhan proyek.
2. Tidak mudah untuk melakukan perbaikan dan pembaharuan (*updating*) *gantt chart* baru yang harus dibuat kembali (tidak efisien). Pembuatan ulang dari *gantt chart* tentunya akan memakan waktu. Jika tidak dilakukan, maka daya guna dari *gantt chart* akan menurun.

Menurut Dana Marsetiya Utama (2019), untuk proyek-proyek yang berukuran sedang dan besar serta kompleks, gantt chart tidak mampu menyajikan jadwal secara sistematis dan mengalami kesulitan dalam menentukan keterkaitan antar kegiatan.

2.10. Nawaz Enscore Ham (NEH)

Metode Nawaz Enscore Ham (NEH) merupakan algoritma heuristik yang dikembangkan oleh Muhammad Nawaz, E. Emory Enscore Jr, dan Inyong Ham pada tahun 1983. Menurut Nawaz, Enscore Jr, and Ham (1983), Cara kerja metode NEH yaitu *job* dengan total waktu proses terbesar diberikan prioritas utama untuk dikerjakan. Inisialisasi urutan *job* NEH secara *descending* berdasarkan total waktu proses setiap *job*. Setelah itu menentukan urutan terbaik dari masing-masing posisi *job*. Menurut penelitian yang dilakukan oleh Park et al. (1984), metode NEH

mengungguli 15 metode heuristic lainnya secara signifikan pada bidang penjadwalan *flow shop*. Langkah-langkah metode NEH:

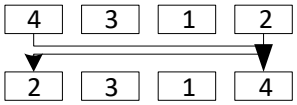
1. Menjumlahkan waktu proses setiap *job*. Selanjutnya, urutkan *job* berdasarkan total waktu proses dari yang terbesar sampai terkecil.
2. Ambil *job* yang menempati urutan pertama dan kedua pada saat pengurutan *job*. Buat 2 alternatif urutan *job* yang baru, kemudian hitung *makespan* masing-masing alternatif tersebut. Dari hasil kedua *makespan* tersebut, pilih hasil *makespan* yang terkecil untuk ditambahkan ke urutan *job* selanjutnya. Jika dari kedua hasil *makespan* yang diperoleh sama, maka pilih salah satu dari alternative tersebut.
3. Selanjutnya ambil *job* yang menempati urutan selanjutnya dari daftar pengurutan *job* awal pada langkah 1. Kemudian, sisipkan *job* tersebut pada urutan *job* yang diperoleh berdasarkan alternatif pada langkah 2. Hitung setiap *makespan* dari masing-masing calon alternatif tersebut. Jika terjadi *makespan* yang dihasilkan sama, maka tetap memilih salah satu dari alternatif tersebut.

2.11. *Local Search*

Menurut Mladenović and Hansen (1997), metode *local search* adalah metode optimasi kombinatorial yang melakukan perubahan urutan solusi awal disetiap iterasi sampai fungsi tujuan yang optimum ditemukan. Artinya pada setiap iterasi, solusi diperbaiki sampai tidak ada perbaikan lebih lanjut ditemukan. Menurut penelitian yang dilakukan oleh Laporte, Gendreau, Potvin, and Semet (2000), menunjukkan bahwa algoritma metaheuristik yang efektif dapat diperoleh dengan melanjutkan perubahan sistematis kedalam *local search*. Langkah *local search* yang digunakan yaitu *do swap*, *do flip*, dan *do slide*. Berikut langkah-langkah *local search*:

1. Lakukan *Do Swap*

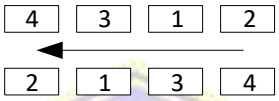
Do swap dilakukan dengan cara menukar dua gen secara acak. Operasi *swap* diulang 1 kali pada setiap iterasi.



Gambar 2. 6 Contoh Do Swap

2. Lakukan *Do Flip*

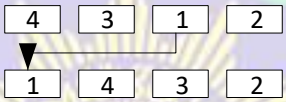
Do Flip dilakukan dengan cara membalikkan urutan gen yang dipilih. Operasi *flip* diulang 1 kali pada setiap iterasi.



Gambar 2. 7 Contoh Do Flip

3. Lakukan *Do Slide*

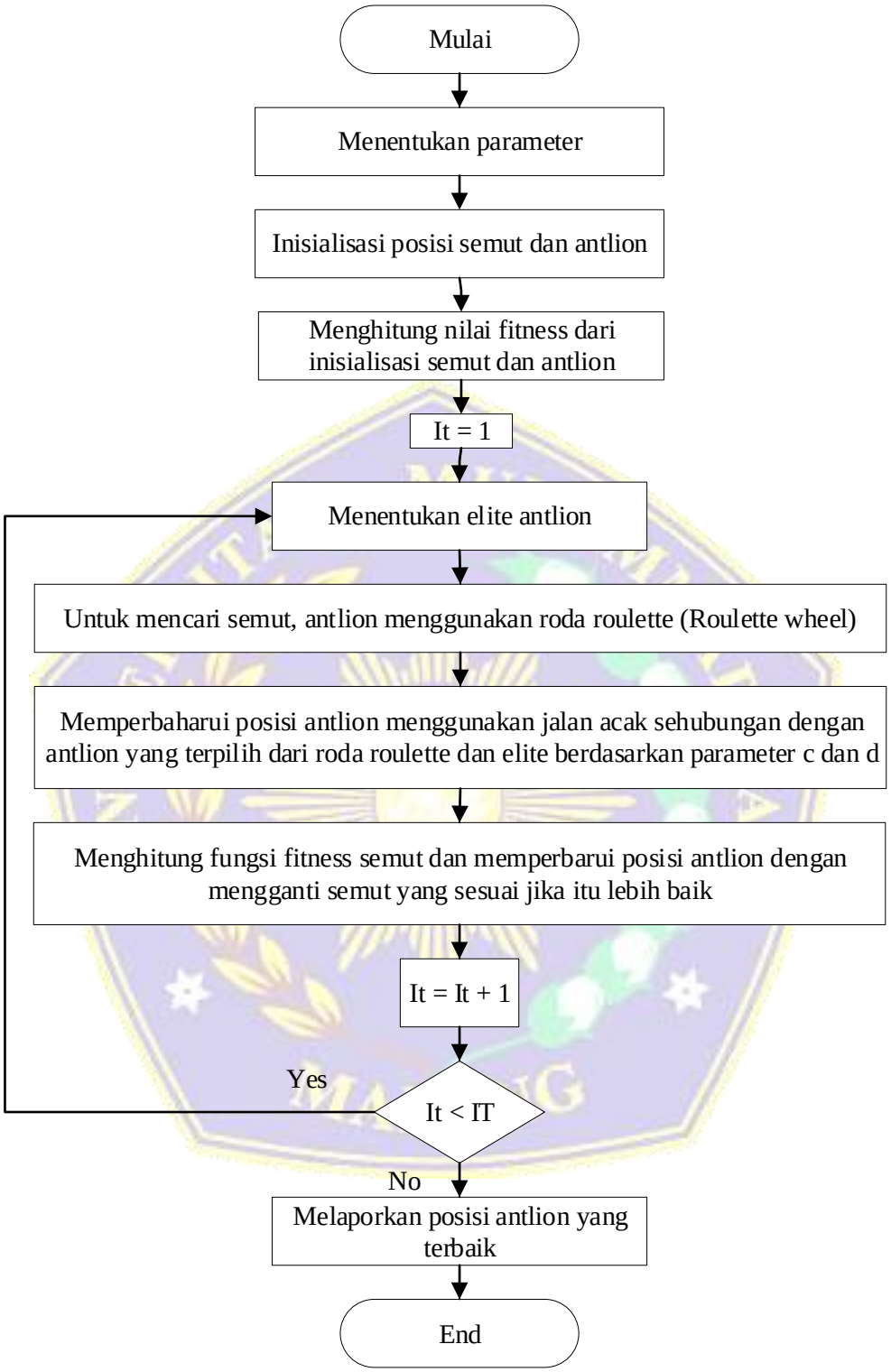
Do slide dilakukan dengan cara menggeser gen. Operasi *slide* diulang 1 kali pada setiap iterasi.



Gambar 2. 8 Contoh Do Slide

2.12. Ant Lion Algorithm (ALO)

Algoritma antlion merupakan algoritma metaheuristic yang terinspirasi oleh kecerdasan perilaku antlion dalam berburu makanan dialam. Menurut penelitian yang dilakukan oleh Mirjalili (2015), pada tahap verifikasi hasil kinerja algoritma ALO ada 2 uji, uji yang dilakukan dibandingkan dengan algoritma lain seperti PSO, algoritma *State of Matter Search* (SMS), *Bat Algorithm* (BA), Algoritma *Cuckoo Search* (CS), dan *Firefly Algorithm* (FA). Pada uji statistic dan Wilcoxon, algoritma ALO lebih stabil dalam memecahkan fungsi tes. Terbukti bahwa hasil yang diperoleh signifikan secara statistik untuk Algoritma ALO. Pada literatur yang diusulkan, algoritma ALO dicoba untuk memecahkan masalah desain baling-baling kapal. Kinerja algoritma ALO bisa menjadi patokan yang efektif dalam memecahkan masalah yang sangat menantang. Menurut Bozorg-Haddad (2018), berikut adalah *flowchart* langkah-langkah dari algoritma ALO pada gambar 2.6.



Gambar 2. 9 Flowchart Algoritma ALO (It = iteration counter ; dan IT = number of iterations)

2.12.1. Inisialisasi Posisi Semut dan Antlion dan Evaluasi Fungsi Kebugaran Mereka

Dalam algoritma ALO, terdapat 2 variabel yaitu semut dan antlion. Semut adalah agen pencarian diruang keputusan dan antlion bersembunyi diruang keputusan. Antlion dapat memburu mereka dan menangkap posisi mereka untuk menjadi bugar. Algoritma ALO dimulai dengan matriks hasil posisi semut dan antlion secara acak dalam rentang yang ditentukan sebagai berikut:

Posisi semut disimpan dalam matriks $P_{Ant} = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & \dots & A_{1,d} \\ A_{2,1} & A_{2,2} & \dots & \dots & A_{2,d} \\ A_{n,1} & A_{n,2} & \dots & \dots & A_{n,d} \end{bmatrix}$ .....(10)

Dimana P_{Ant} : matriks posisi semut, $A_{n,d}$: d adalah variabel keputusan semut ke-n, n adalah jumlah semut.

Posisi semut disimpan dalam matriks $P_{Antlion} = \begin{bmatrix} AL_{1,1} & AL_{1,2} & \dots & \dots & AL_{1,d} \\ AL_{2,1} & AL_{2,2} & \dots & \dots & AL_{2,d} \\ AL_{n,1} & AL_{n,2} & \dots & \dots & AL_{n,d} \end{bmatrix}$.(11)

Dimana $P_{Antlion}$: matriks posisi Antlion, $A_{n,d}$: d adalah variabel keputusan antlion ke-n, n adalah jumlah Antlion.

Untuk mengevaluasi semut dan antlion, fungsi kebugaran dari mereka digunakan dan nilai kebugarannya dihitung selama optimasi, kemudian disimpan dalam matriks berikut. Dalam proses ini, antlion dengan kebugaran terbaik dipilih sebagai elit.

$$F_{ant} = \begin{Bmatrix} f\left(\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,5} & \cdots & A_{1,d} \end{bmatrix}\right) \\ f\left(\begin{bmatrix} A_{2,1} & A_{2,2} & \cdots & A_{2,5} & \cdots & A_{2,d} \end{bmatrix}\right) \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ f\left(\begin{bmatrix} A_{5,1} & A_{5,2} & \cdots & A_{5,5} & \cdots & A_{5,d} \end{bmatrix}\right) \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ f\left(\begin{bmatrix} A_{n,1} & A_{n,2} & \cdots & A_{n,5} & \cdots & A_{n,d} \end{bmatrix}\right) \end{Bmatrix} \text{.....(12)}$$

Dimana F_{ant} adalah matriks fungsi kebugaran semut, dan f adalah fungsi kebugaran.

$$F_{antlion} = \begin{Bmatrix} f\left(\left[Al_{1,1}\ Al_{1,2}\ \cdots\ Al_{1,5}\ \cdots\ Al_{1,d}\right]\right) \\ f\left(\left[Al_{2,1}\ Al_{2,2}\ \cdots\ Al_{2,5}\ \cdots\ Al_{2,d}\right]\right) \\ \vdots\ \ \vdots\ \ \vdots\ \ \vdots\ \ \vdots\ \ \vdots \\ f\left(\left[Al_{5,1}\ Al_{5,2}\ \cdots\ Al_{5,5}\ \cdots\ Al_{5,d}\right]\right) \\ \vdots\ \ \vdots\ \ \vdots\ \ \vdots\ \ \vdots\ \ \vdots \\ f\left(\left[Al_{n,1}\ Al_{n,2}\ \cdots\ Al_{n,5}\ \cdots\ Al_{n,d}\right]\right) \end{Bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(13)$$

Dimana $F_{antlion}$ adalah matriks fungsi kebugaran antlion.

2.12.2. Menggali Perangkap

Algoritma ALO perlu menggunakan roda roulette untuk memilih antlion sesuai dengan kebugaran mereka selama optimasi. Mekanisme ini memberi peluang tinggi bagi antlions yang lebih bugar untuk menangkap semut dengan membangun perangkap. Perlu dicatat bahwa dalam algoritma ALO, setiap semut dapat jatuh kedalam hanya pada satu perangkap antlion di setiap iterasi. Roda roulette yang dipilih sebagai antlion untuk setiap semut adalah antlion yang telah menjebak semut. Dengan menggunakan roda roulette, solusi fungsi kebugaran yang baik memiliki lebih banyak kesempatan untuk dipilih, karena antlion dengan perangkap yang lebih besar dapat memburu lebih banyak semut.

2.12.3. Geser Semut Menuju Antlion

Ketika semut jatuh kedalam jebakan, antlion mulai menembakkan pasir keluar untuk meluncurkan semut kebawah saat semut sedang berusaha melarikan diri. Perilaku ini dimodelkan secara matematis dengan mengecilkan jari-jari jalan semut secara acak. Jadi kisaran batas untuk semua variabel keputusan dikurangi dan diperbarui, seperti yang dinyatakan dalam Persamaan. (14) dan (15) (Mirjalili, 2015).

$$\gamma(It) = \frac{c(It)}{R} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(14)$$

$$\delta(It) = \frac{d(It)}{R} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(15)$$

Dimana $\gamma(It)$ = vektor yang dimodifikasi termasuk minimum semua variabel keputusan pada iterasi Itth; $c(It)$ = vektor yang termasuk minimum semua variabel keputusan pada iterasi Itth; R = rasio yang diberikan oleh Persamaan. (16); $\delta(It)$ = vektor yang dimodifikasi termasuk maksimum semua variabel

keputusan pada iterasi Itth; dan d(It) = vektor termasuk maksimum semua variabel keputusan pada iterasi Itth (Mirjalili, 2015).

$R = 10^{w} \frac{It}{IT}$……..……(16)

Dimana w = konstanta didefinisikan berdasarkan nomor iterasi yang diberikan oleh:

$$W = \begin{cases} 2 \; If \; It > 0{,}1IT \\ 3 \; If \; It > 0{,}5IT \\ 4 \; If \; It > 0{,}75IT \\ 5 \; If \; It > 0{,}9IT \\ 6 \; If \; It > 0{,}95IT \end{cases} \quad \text{……..(17)}$$

Ketika nomor iterasi dalam Persamaan (17) meningkat, jari-jari jalan acak menurun, yang menjamin konvergensi algoritma ALO.

2.12.4. Menjebak Semut Kedalam Lubang

Perangkap antlion mempengaruhi jalan acak semut. Untuk memodelkan perilaku ini secara matematis, batas dari jalan acak semut disesuaikan pada setiap iterasi. Sehingga semut bergerak dalam hyper-sphere di sekitar perangkap antlion yang terpilih. Batas bawah dan atas dari jalan acak semut untuk setiap dimensi dalam setiap iterasi dapat dihitung dengan persamaan berikut (Mirjalili 2015):

γ_m(It) = $Antlion_l$(It) + γ(It)……..(18)

δ_m(It) = $Antlion_l$(It) + δ(It)……..(19)

Dimana γ_m(It) = vektor termasuk minimum semua variabel keputusan untuk semut dalam iterasi Itth; $Antlion_l$(It) = posisi antlion yang terpilih dalam iterasi Itth; dan δ_m(It) = vektor termasuk minimum semua variabel keputusan untuk semut m. Persamaan (18) dan (19) menunjukkan bahwa berjalan semut secara acak berada di hyper-sphere, yang didefinisikan oleh vektor γ dan δ di sekitar roda roulette antlion yang terpilih.

2.12.5. Jalan Acak Semut

Pada langkah ini, semut bergerak secara acak dialam untuk makan, berjalan secara acak digunakan untuk memodelkan gerakan mereka, yang dapat dinyatakan sebagai persamaan berikut (Mirjalili, 2015):

$$X = [0, \text{cumsum}(2r(1)-1), \text{cumsum}(2r(2)-1), \dots, \text{cumsum}(2r(It)-1), \dots, \text{cumsum}(2r(IT)-1)] \quad \text{.....(20)}$$

Dimana X = vektor posisi jalan acak; cumsum = jumlah kumulatif; dan r(It) = fungsi stokastik yang dihitung dengan:

$$r(It) = \begin{cases} 1 \text{ jika rand} > 0{,}5 \\ 0 \text{ otherwise} \end{cases} \quad \text{.....(21)}$$

Dimana rand = nilai acak yang dihasilkan dengan distribusi yang seragam antara 0 dan 1.

Untuk menjaga jalan acak dalam ruang keputusan, mereka dinormalisasi dengan metode normalisasi minimum (Mirjalili, 2015):

$$Z_n(It) = \frac{(X_n(It) - \alpha_n) \times (\delta_n(It) - \gamma_n(It))}{(\beta_n - \alpha_n)} \quad \text{.....(22)}$$

Dimana Z_n(It) = posisi random walk yang dinormalisasi dari variabel keputusan ke-n pada iterasi Itth; X_n (It) = posisi walk random dari variabel keputusan ke-n pada iterasi Itth sebelum normalisasi; α_n = minimum jalan acak untuk variabel keputusan ke-n; β_n = maksimum jalan acak untuk variabel keputusan ke-n; γ_n(It) = minimum dari variabel keputusan n pada iterasi Itth; dan δ_n(It) = maksimum dari variabel keputusan n pada iterasi Itth.

2.12.6. Elitisme

Elitisme adalah karakteristik penting dari algoritma ALO. Elitisme digunakan untuk mempertahankan solusi terbaik yang diperoleh pada setiap tahap iterasi. Pada penelitian ini antlion terbaik yang telah diperoleh pada setiap iterasi disimpan dan dianggap sebagai elit. Antlion elit merupakan antlion yang paling kuat dan dapat mempengaruhi pergerakan semua semut selama iterasi. Oleh karena itu, diasumsikan bahwa setiap semut berjalan acak disekitar antlion yang dipilih berdasarkan roda roulette dan elit secara bersamaan. Berikut persamaan 23 posisi semut:

$$Ant_m^{It} = \frac{R_l^{It} + R_e^{It}}{2} \quad \text{.....(23)}$$

Dimana Ant_m^{It} = posisi antlion yang dipilih dalam It iterasi ; R_l^{It} = berjalan secara acak disekitar antlion yang terpilih dari roda roulette pada iterasi Itth; dan R_e^{It} = berjalan acak di sekitar antlion elit diiterasi Itth.

2.12.7. Menangkap Mangsa dan Memperbaiki Perangkap

Langkah terakhir dalam perburuan, seekor semut jatuh kedasar perangkap dan ditangkap oleh rahang antlion. Antlion menarik semut kedalam pasir dengan rahangnya dan kemudian mengkonsumsinya. Dalam algoritma ALO, menangkap mangsa terjadi ketika fungsi kebugaran semut menjadi lebih baik daripada antlion. Pada situasi ini, antlion mengubah posisinya keposisi yang diburu. Proses seperti ini dinyatakan dengan persamaan berikut:

$Antlion_l^{It} = Ant_m^{It}$ if $f(Ant_m^{It}) > f(Antlion_l^{It})$ .....(24)

2.13. *Literature Review* Jurnal Pendukung Penelitian

Beberapa penelitian terdahulu tentang penjadwalan *flowshop* meliputi Petrović et al. (2016) dalam penelitiannya menunjukkan bahwa *Algoritma Ant Lion Optimization* (ALO) dibandingkan dengan Algoritma Genetika, diperoleh bahwa ALO memberikan nilai *makespan* yang kecil, Ali, Elazim, and Abdelaziz (2018) dalam penelitiannya algoritma *Antlion Optimization* (ALO) diusulkan untuk pengoptimalan lokasi dan ukuran dari *Distributed Generation* (DG) yang berdasarkan sumber terbaru untuk system distribusi radial. Hasil yang diperoleh melalui algoritma ALO dibandingkan dengan lainnya untuk menyoroti manfaatnya dalam mengurangi kehilangan daya total dan akibatnya dapat memaksimalkan penghematan, Tasgetiren, Kizilay, Pan, and Suganthan (2017) menunjukkan algoritma *Iterated Greedy* (IG) dalam memecahkan masalah penjadwalan *blocking flowshop* dengan kriteria minimasi *makespan*. Kinerja algoritma IG secara signifikan tergantung pada metode percepatan yang digunakan. Hasil komputasi dari penelitian ini menunjukkan bahwa algoritma IG yang diulang dengan metode percepatan pada rangkaian Taillard akan lebih unggul dari algoritma lain pada literature ini., Bargaoui, Belkahla Driss and Ghédira (2017) dalam penelitiannya menunjukkan algoritma *Chemical Reaction Optimization* (CRO) dalam menyelesaikan masalah penjadwalan permutasi *flowshop* untuk meminimasi *makespan*. Pada hasil simulasi, algoritma CRO mampu menghasilkan solusi yang lebih baik dibandingkan dengan *Variable Neighborhood Descent heuristics* (VND), dan *NEH heuristics*, Lu, Gao, Pan, Li, and Zheng (2019) mempresentasikan algoritma *Multi-Objective Cellular Grey Wolf Optimizer* (MOCGWO) untuk

masalah penjadwalan *hybrid flowshop* (HFSP) dengan mempertimbangkan polusi suara/ kebisingan. Tujuannya adalah untuk mewujudkan penjadwalan hijau dengan pengurangan terhadap kebisingan. Untuk dapat menunjukkan efektivitas algoritma ini dalam mengatasi HFSP, maka dilakukan percobaan perbandingan pada 20 contoh uji. Hasilnya MOCGWO lebih unggul dibandingkan *Multi-Objective Evolutionary Algorithms* (MOEA) dalam masalah ini, Wang et al. (2017) menunjukkan algoritma Cuckoo Search (CS) yang disebut New Cuckoo Search (NCS) untuk memecahkan masalah penjadwalan flowshop (FSSP). NCS menghibridasi 4 strategi: (1) untuk menangani variable diskrit, (2) untuk menghasilkan solusi awal berkualitas tinggi, (3) untuk meningkatkan eksploitasi, (4) untuk mempercepat konvergensi. Hasil percobaan menunjukkan bahwa NCS memperoleh kinerja yang lebih baik daripada CS dan beberapa algoritma meta-heuristik lainnya seperti A Novel PSO (NPSO), Improved Hybrid CS (IHCS), dll, Bewoor, Chandraprakash, and Sapkal (2018) dalam penelitiannya mengusulkan hibridasi *Particle Swarm Optimization* (PSO) dengan *Simulated Annealing* (SA) untuk penjadwalan *flowshop* tanpa adanya waktu tunggu. Algoritma dikembangkan tidak hanya menerapkan pencarian evolusioner berbasis PSO, tetapi juga menggunakan pencarian local berbasis SA dengan populasi awalnya berbasis NEH. Oleh karena itu, keseimbangan eksploitasi dan eksplorasi yang tepat dapat dipastikan secara global. Hasil dari perbandingan berbagai set data yang digunakan untuk simulasi menunjukkan bahwa algoritma ini unggul dalam hal kualitas pencarian dan ketahanan solusi, G. Komaki, Teymourian, and Kayvanfar (2016) pada penelitiannya menunjukkan masalah penjadwalan *hybrid flowshop scheduling* (HFS) dengan operasi perakitan. Dimana beberapa produk pada operasi perakitan harus diproduksi sendiri. Sehingga setiap produk memiliki *Bill of Material* (BOM). Algoritma *Hybrid Artificial Immune Systems* diusulkan pada penelitian ini untuk menemukan urutan produk, serta urutan bagian dari setiap produk secara bersamaan. Tujuan dari penelitian yang dilakukan adalah untuk mendapatkan hasil *makespan* yang minimum, Abdollahpour and Rezaeian (2015) mengusulkan tiga algoritma metaheuristic baru yaitu algoritma *artificial immune system* (AIS), algoritma *iterated greedy* (IG), dan *hybrid artificial immune system* (AIS-IG).

Performansi yang digunakan adalah untuk meminimasi *makespan*. Berbeda dari AIS dasar, algoritma AIS-IG dikombinasikan dengan fase penghancuran dan algoritma IG untuk meningkatkan eksplorasi lokal. Pada hasil penelitian, terbukti algoritma AIS-IG dapat mempertahankan kualitasnya untuk masalah dengan skala besar.

Tabel 2. 1 *Literatur Review* Jurnal Pendukung

Nama Penulis	**Jenis Lingkungan Manufaktur**	**Metode**	**Performansi**
Costa, Cappadonna, and Fichera (2017)	*Flowshop*	*Hybrid Genetic Algorithm*	Minimasi *Makespan*
Cui and Gu (2015)	*Hybrid Flowshop*	*Improved Discrete Artificial Bee Colony* (IDABC)	Minimasi *Makespan*
Han, Gong, Li, and Zhang (2016)	*Blocking Flowshop*	*Modified Fruit Fly Optimization* (MFFO) *Algorithm*	Minimasi *Makespan*
Mansouri, Aktas, and Besikci (2016)	*Flowshop*	*Green Scheduling*	Minimasi *Makespan* dan Konsumsi Energi
M. Komaki and Malakooti (2017)	*Flowshop*	*General Variable Neighbourhood Search* (GVNS) *Algorithm*	Minimasi *Makespan*
Shao, Pi, and Shao (2017)	*Flowshop*	*Memetic Algorithm*	Minimasi *Makespan*
Bargaoui, Driss, and Ghédira (2016)	*Flowshop*	*Chemical Reaction Optimization*	Minimasi *Makespan*
Sanjeev Kumar, Padmanaban, and Rajkumar (2018)	*Permutation Flowshop*	*Gravitational Emulation Local Search*	Minimasi *Makespan* dan Total *Flowtime*
Chen, Wen, Li, and Li (2017)	*Permutation Flowshop*	*Hybrid Backtracking Search Algorithm*	Minimasi *Makespan* dan Konsumsi Energi